What Is Claimed Is:

2

3

4

| 1 | 1. A method for executing a start transactional execution (STE) |
|----|---|
| 2 | instruction to facilitate transactional execution on a processor, comprising: |
| 3 | encountering the STE instruction during execution of a program, wherein |
| 4 | the STE instruction marks the beginning of a block of instructions to be executed |
| 5 | transactionally; and |
| 6 | upon encountering the STE instruction, commencing transactional |
| 7 | execution of the block of instructions following the STE instruction; |
| 8 | wherein changes made during the transactional execution are not |
| 9 | committed to the architectural state of the processor until the transactional |
| 10 | execution successfully completes. |
| | |
| 1 | 2. The method of claim 1, wherein the STE instruction specifies an |
| 2 | action to take if transactional execution of the block of instructions fails. |
| | |
| 1 | 3. The method of claim 2, wherein the action to take can include |
| 2 | branching to a location specified by the STE instruction. |
| | |
| 1 | 4. The method of claim 2, wherein the action to take can include |
| 2 | acquiring a lock on the block of instructions. |
| | |
| 1 | 5. The method of claim 2, wherein the action to take can include |

setting state information within the processor to indicate a failure during

software executed by the processor to manage the failure.

transactional execution of the block of instructions, thereby enabling other

| 1 | o. The method of claim 1, wherein if the transactional execution |
|---|--|
| 2 | completes without encountering an interfering data access from another process o |
| 3 | other type of failure, the method further comprises: |
| 4 | atomically committing changes made during the transactional execution, |
| 5 | and |
| 6 | resuming normal non-transactional execution. |
| 1 | 7. The method of claim 1, wherein if an interfering data access from |
| 2 | another process is encountered during the transactional execution, the method |
| 3 | further comprises: |
| 4 | discarding changes made during the transactional execution; and |
| 5 | attempting to re-execute the block of instructions. |
| 1 | 8. The method of claim 1, wherein potentially interfering data |
| 2 | accesses from other processes are allowed to proceed during the transactional |
| 3 | execution of the block of instructions. |
| 1 | 9. The method of claim 1, wherein the block of instructions to be |
| 2 | executed transactionally comprises a critical section. |
| 1 | 10. The method of claim 1, wherein commencing transactional |
| 2 | execution of the block of instructions involves: |
| 3 | saving the state of processor registers; |
| 4 | configuring the processor to mark cache lines during loads that take place |
| 5 | during transactional execution; |
| 6 | configuring the processor to mark cache lines during stores that take place |
| 7 | during transactional execution; and |

| 9 | other threads to detect interfering data references. |
|----|--|
| 1 | 11. The method of claim 1, wherein the STE instruction is a native |
| 2 | machine code instruction of the processor. |
| 1 | 12. The method of claim 1, wherein the STE instruction is defined in a |
| 2 | platform-independent programming language. |
| 1 | 13. A computer system that supports a start transactional execution |
| 2 | (STE) instruction to facilitate transactional execution, wherein the STE instruction |
| 3 | marks the beginning of a block of instructions to be executed transactionally, the |
| 4 | computer system comprising: |
| 5 | a processor; and |
| 6 | an execution mechanism within the processor; |
| 7 | wherein upon encountering the STE instruction, the execution mechanism |
| 8 | is configured to commence transactional execution of the block of instructions |
| 9 | following the STE instruction; |
| 10 | wherein changes made during the transactional execution are not |
| 11 | committed to the architectural state of the processor until the transactional |
| 12 | execution successfully completes. |
| 1 | 14. The computer system of claim 13, wherein the STE instruction |
| 2 | specifies an action to take if transactional execution of the block of instructions |
| 3 | fails. |

configuring the processor to continually monitor data references from

8

| 1 | 15. The computer system of claim 14, wherein the action to take can |
|---|---|
| 2 | include branching to a location specified by the STE instruction. |
| | |
| 1 | 16. The computer system of claim 14, wherein the action to take can |
| 2 | include acquiring a lock on the block of instructions. |
| | |
| 1 | 17. The computer system of claim 14, wherein the action to take can |
| 2 | include setting state information within the processor to indicate a failure during |
| 3 | transactional execution of the block of instructions, thereby enabling other |
| 4 | software executed by the processor to manage the failure. |
| | |
| 1 | 18. The computer system of claim 13, wherein if the transactional |
| 2 | execution completes without encountering an interfering data access from another |
| 3 | process or other type of failure, the execution mechanism is configured to: |
| 4 | atomically commit changes made during the transactional execution, and |
| 5 | to |
| 6 | resume normal non-transactional execution. |
| | |
| 1 | 19. The computer system of claim 13, wherein if an interfering data |
| 2 | access from another process is encountered during the transactional execution, the |
| 3 | execution mechanism is configured to: |
| 4 | discard changes made during the transactional execution; and to |
| 5 | attempt to re-execute the block of instructions. |
| | |
| 1 | 20. The computer system of claim 13, wherein the computer system is |
| 2 | configured to allow potentially interfering data accesses from other processes to |
| 3 | proceed during the transactional execution of the block of instructions. |

| 1 | 21. The computer system of claim 13, wherein the block of |
|----|--|
| 2 | instructions to be executed transactionally comprises a critical section. |
| | |
| 1 | 22. The computer system of claim 13, wherein while commencing |
| 2 | transactional execution of the block of instructions, the execution mechanism is |
| 3 | configured to: |
| 4 | save the state of processor registers; |
| 5 | configure the processor to mark cache lines during loads that take place |
| 6 | during transactional execution; |
| 7 | configure the processor to mark cache lines during stores that take place |
| 8 | during transactional execution; and to |
| 9 | configure the processor to continually monitor data references from other |
| 10 | threads to detect interfering data references. |
| | |
| 1 | 23. The computer system of claim 13, wherein the STE instruction is a |
| 2 | native machine code instruction of the processor. |
| | |
| 1 | 24. The computer system of claim 13, wherein the STE instruction is |
| 2 | defined in a platform-independent programming language. |
| | |
| 1 | 25. A computing means that supports a start transactional execution |
| 2 | (STE) instruction to facilitate transactional execution, wherein the STE instruction |
| 3 | marks the beginning of a block of instructions to be executed transactionally, |
| 4 | comprising: |
| 5 | a processing means; and |
| 6 | an execution means within the processing means; |

| 7 | wherein upon encountering the STE instruction, the execution means is |
|----|---|
| 8 | configured to commence transactional execution of the block of instructions |
| 9 | following the STE instruction; |
| 10 | wherein changes made during the transactional execution are not |
| 11 | committed to the architectural state of the processor until the transactional |
| 12 | execution successfully completes. |